CSP - Assignment

AP19110010351
CSE - F

1.

```
#include <stdio.h>
# include   < stdlib.h>
  void insert (Node*, int, int)
   int size=0;
   struct node {
    int data;
    struct node * next;
    };
  Node*   get-node (int data)
 {
   Node *  newnode = (struct node*)malloc
                                      (newnode);
        newnode -> data = data;
        newnode -> next = NULL;
        return newnode;
 }
  void insert (node * current, int position, int data)
 {
   if (Pos < -1 // Pos > size+1)
      Printf ("Invalid");
    else {
      while (Pos--)
     {
       if (Pos = =0)
      {
        Node * temp = getnode (data);
        temp -> next =* current;
       *current = temp;
      }
      else {
```

```
Current  =  & ( * current)  -> next;
  }
   size ++;
   }
   }
    void print (struct node * head)
    {
     while (head != NULL)
     {
      Printf ("%d", head -> data);
        head = head -> next;
      }
      Printf ("/n");
      }
      void del (struct node * head - ref, int position)
      {
        if  (head - ref = NULL)
             return;
          temp = head - ref :
           if (Pos = 0)
         {
         head - ref = temp -> next;
           free (temp);
          return;
   for (int i = 0; temp != NULL && i < Pos-1; i ++)
        temp = temp -> next;
         free (temp -> next);
           temp -> next = next;
        }
         int main ( )
```

```
{
   struct node * head = NULL;
     Push (&head, 12);
     Push (&head, 15);
     Push (&head, 8);
     insert (&head, 4, 9, 3);
      delete (&head, 12);
      return 0;

   }
```

step 1: statement of the program

construct a new linked list by merging alternate nodes of two lists for example in list 1 we have {1,2,3} and in list 2 we have {4,5,1} in the new list we should have {1,4,2,5,3,6}

step2: Explaination of the program

Here first we should create two new linked list then we should merge alternate nodes of second linked list with first linked list.

step3: steps and Algorithm involved in the program.

1) create a structure.

2) Function to insert a node at beginning

3) Function to print singly linked list

4) Function that inserts nodes of linked q into P alternate position.

5) Program to test the above function.

step4 - Code in C language

```
# include <stdio.h>
  # include <stdlib.h>
    struct Node {
        int data;
        struct Node * next;

    };
    void push (struct Node** head_ref, int
                                          new_data)
    {
      struct Node* new_node = (struct Node*)
      malloc(size of (struct Node));
         new_node -> data = new_data;
         new_node -> next = (*head_ref);
         (*head_ref) = new_node;
    }
       void print List (struct Node* head)
     {
        struct Node* temp = head;
        while (temp! = NULL)
      {
         Print F ("%d", temp->data);
            temp = temp->next;
      }
         print f ("\n");
      }
  void merge (struct Node * a, struct Node ** b)
    {
      struct Node* a_first = a, * v_first =* v;
        struct Node* a_next, * v_next;
         while (a_first ! = NULL && v_first!
           = NULL)
    {
         a_next = a_first->next;
          v_next = v_first->next;
         v_first ->next = a_next;
         a_first ->next = v_first;

        a_first = a_next;
          v_first = v_next;

    }

        *q = v_first;

  }
int main ()
{ struct Node * p= NuLL, * q = NULL;
     Push(&a, 3);
      Push(&a, 2);
      Push (&a, 1);
      printf ("First linked list a :\n"
        Print List (a);
      Push (&v, 8);
      Push (&v, 7);
      Push (&v, 6);
    Push (&v, 5);
     Push (&v, 4);
  Printf(" second linked list v:\n");

      Print list (v)
        merge (a, &v);
        Print f(" Merged first Linked list a:\n");
        PrintList (a);
       Printf(" Modified second linked list v:\n");
        Print list (v);
           get char ();
             return 0;

      }
```

3.

```
#include <stdio.h>
  void find (int ar[100], int n, int r)
 {
   int sum 0;
    int i=0, h=0;
    for (i=0; i<n; i++)
  {
    while ((sum < r) && (h<n))
    {
      sum = sum + ar[h];
        h++;
       if (sum = r)
     { Printf("Found");
       return;
}
sum = ar[l];
}
}
int main (void)
{ int ar{100} ={3,4,7,9,8}
  int s =25;
  find (ar, n, s);
  return 0;
  }
```

4.
(i)

```
#include <stdio.h>
# include <stdlib.h>
struct stackrecord
{ int * array ;
  int capacity;
  int top;
  };
  typedef struct stackrecord * stack;
  stack (createstack (int max);
  {
   stack s;
   s = malloc (size of (struct stackrecord));
   if (s == NULL)
  {
```

```
Print f (" out of space");
 }
  s->array =malloc ((size of (int))*Max);
  if (s -> array = = NULL)
  {
   print f ("out of space");
   }
    s->capacity = Max - 1;
    s -> tos = -1
 return (s);

  }
  int is empty (stack s)
  {
   return . s - tos = -1;
     }
     int is full (stack s)
     return s -> to s = s -> capacity;
     }
      void push (int x, stack s)
    {  if (is fulle (s))

     Print f ("over flow");
     else
  {  print f ("\n %d is pushed", x);
      s -> tos++;
      s -> array [s ->tos] = x;
   }
```

```
int top up and pop (stack s)
 { if (is empty (s))
   { Print f (" \n empty queue");
      return 0;
    }
     else
     {
      Print f (" \n %. d is popped", s->array
                                      [s - tos]);
      return s->array (s->tos--]);
       }
       }
 if (is full q (q))
    Print f (" over flow ");
    else
   {
    Print f (" /n %.d is enqueued", x);
     q->rear++;
    q->array. [q->rear] = x;
     if (q->front = = -1)
      q->front ++;
      }
      }
       int front and delete (queue q)
   { int x;
     if (is empty q (q))
   {
```

```
print f ("underflow");
    return 0;

   }
   else
  {
  p = q -> array[q->front];
  printf ("\n%d is front and delete", p);
      q->front++;
         return p;

            }
            }
         void display (queue q)
            {
             int i, rear;
          if (isemptyq(q))
          {
  struct queuerecord
 {
  int .* arry;
  int front;
  int rear;
  int capacity.

  } ;
 typedef struct quee card * que;
  quee create quee (int max)

 {
  quee q;
  q = malloc (size of (struct queuecard))
if (a = = NULL)
 Printf ("Error");
   q -> array = malloc (size of (int)* max);
     if (q-> array = = NULL)
       Printf ("Error");
        q -> capacity = max -1;
         q -> front = -1;
          q -> real = -1;
          return q;
         }
         int is full q (queue q)
        {
          return (q-> real = = q -> capacity);
        } int is empty q (queue q)
          {
            return (q -> front = = -1);
          }
          void enqueue (queue q, int x)
          {
     Printf ("underflow");
      return;
       }
        for (i = q -> front; i <= real; i++)
        Printf ("%d \t", q -> array [i]);
       }
       int main()
      {
      int max, ch, i, choice, n = 0, 1, 2, capacity
        queue q;
```

```
clrscr;
printf("\n Enter the maximum elements");
   scanf("%d", &max);
      q = create queue (max);
      s = create stack (max);
       while (1)

   {
     printf("\n menu : 1. Insert 2. Display
                reversed order . 3 exit");
   printf ("Enter the choice");
     scanf ("%d", &choice);
        switch (choice)
        {
         case 1:
           printf("\n Enter the element");
           scanf("%d", &ele);
           Enqueue (q, ele);
         n++;
         break;
         case 2;
         printf("\n contents of the
               queue : ");
          display (q);
          for (i = 0; i<capacity; i++)
  t = front and delete (q);
     Push (t, s);
 }
q -> front = -1;
 q->rear = -1;
     for (i = 0; i<capacity; i++)
    {
    Y = top and pop (s);
       enque (q, v);
      }
       printf (" \n   Reversed (ontents
                          are :");
       display (q);
           to made;
         Case 3;
        exit (0);
      }
      }
      }
      }
```

(ii)
```
#include<stdio.h>
    #include <stdlib.h>
       struct node
       {
         int data;
         struct node* next;
          };
         void push ((struct node *head
       {                  ref, char new).
```

```
struct node * node_new = (struct node *)
    malloc (size of (struct node));
            node ~~new~~ -> data = new;
        node - new -> next = (head_ref;
          (* head - ref ) = node - new
    }
      int main ()
      {
        struct node* head = NULL;
                  Push (&head, 7);
                  Push (&head, 5);
                  Push (&head, 3);
                  Push (&head 1);
                  Push (&head 8);
              Print alternate (head);
                  return 0;
  }
  void print alternate (struct node* head
        {
         int count = 0;
          while (head != NULL)
        {
  if (count % 2 == 0) {
     count << head -> data << " ";
           count ++;
          head = head -> next;
     }}}
```

5. (i) An array is the data structure that contains a collection of similar type data elements whereas the linked list s considered as non-permitive data structure contains a collection of unordered linked elements known as node.

In a linked list though, you have to start from the head and work your way through until you get to the fourth element.

Operations like insertion and deletion in arrays consume a lot of time. On the other hand, the performance of these operations in linked lists is fast.

In an array, memory is assigned during the compile time while in a linked list it is allocated during execution or ~~run~~ runtime.

Elements are stored consecutively in arrays whereas it is stored randomly in linked lists

In addition memory utilization is inefficient in the array ~~conversely~~, conversely memory utilization is efficient

```
(ii) #include <stdio.h>
      int main ()
    {
      int a[100], b[100];
      int i, v, position, n=10;
         for (i=0; i<10; i++)
       {
         scanf ("%d", &a,[i]);
         } for (i=0; i<n; i++)
            printf ("%d", b[1]);
           printf ("%d", b[2]);
              v = b2[0];
                 position = 1;
                n++;
                 for (i=n; i>=position; i--
                      b1[i] = b1[1-1];
                        b[position-1] = v;
                  for (i=0; i<n; i++).
                       printf ("%d", b1[i]);
                      for (i=1; i<n; i++)
                         printf ("%d", b2[i]
                         return 0;
                           }
```